Result and Output of AirportRouteFinder

There are three main goals in our project:

- **Build a graph base on the airports and routes data**
  - We build a graph structure having airports as Vertices and routes as Edges.
  - The graph built the foundation of BFS, Dijkstra, and A* algorithm.
- **Use BFS traversal to find destinations after a number of transfers**
  - We use BFS to find destinations based on the given starting location and the number of times we would like to transfer.
    - input: `starting point, number of times we want to transfer`
    - output: name of airports we can reach within the number of transfers
    - Suppose we are given the starting point of CMI, set the number of transfers to 1 and 2, we can get the result as following:

```
Total airports we can reach after 0 times of transfer: CMI, ORD, DFW
Total airports we can reach after 1 times of transfer: CMI, ORD, DFW,
YXE, SXM, STC, SRQ, SPI, SJO, SCE, CDG, SBN, SAV, SGF, ORF, AVL, YYC,
OGG, MKG, LAN, MHK, CAK, JAN, MSN, IAD, YQR, CAE, GSO, RAP, SMF, ELM,
GSP, GRU, FOE, EAU, CMX, CHS, ABQ, BOI, AVP, ALB, BRU, BDL, ICN, CPH,
ARN, OAK, MYR, LNK, HNL, WAW, ABE, SAT, MUC, BWI, PUJ, AMS, DEL, ROA,
YQB, BHM, YEG, ACY, YUL, ALO, PAH, YKF, PVG, XNA, TXL, YHZ, TUS, TUL,
AUS, TPA, MHT, TOL, MLI, SYR, DEN, EWR, SUX, STL, SJU, SEA, SAN, VIE,
CLT, PTY, RST, STT, ROC, COU, RNO, RDU, PVR, PHL, PBI, MSY, YVR, MQT,
MIA, CLE, MEX, DOH, MEM, LSE, MDT, LGA, EVV, LEX, SLC, CWA, LAS, JAX,
PNS, DLH, PIT, RIC, IND, ICT, FSD, MAN, MTY, HPN, MCO, LIT, FWA, JFK,
RSW, FLL, FAR, PHX, FRA, DBQ, YYZ, DUS, IAH, BTV, DUB, SJC, FNT, LAX,
MCI, DTW, ELP, PDX, HSV, GCM, OMA, ATW, NRT, DCA, HKG, SAL, OKC, YWG,
NAS, ZRH, GDL, LHR, YXU, CVG, SFO, PSP, BZN, CUN, YOW, FCO, CMH, BUF,
CHA, CHO, MAD, CID, MOB, PEK, BOS, SNA, MKE, DAY, SJD, BMI, SDF, AZO,
COS, ATL, DEC, AUH, PVD, GRR, MBS, IST, ANC, GRB, TYS, BRL, PIA, BNA,
TVC, ART, CRW, MSP, DSM, PWM, MBJ, AMM, DXB, TLH, SPS, SLP, SJT, SHV,
SCL, RTB, ROW, ZCL, QRO, TXK, PLS, VPS, ONT, MZT, MLU, MGM, MAF, LIM,
LFT, ACT, LCH, LAW, JLN, TYR, LBB, DRO, GRK, HOU, GUA, GGG, GJT, TRC,
GCK, BJX, GIG, BTR, GRI, FSM, AMA, MLM, CUU, LRD, AEX, GPT, CRP, BRO,
CLL, MFE, EZE, CCS, BZE, BPT, FAT, BOG, SAF, LIR, ABI, PBC, CZM, AGU,
BNE
```

- **Find preferred (shortest) route between two recorded airports with possible routes given under certain conditions**
  - We dealt with the following **exceptions**:
    - Input airports do not exist
      - resulting output: `Invalid Input!`
    - The starting airport is the same as the destination airport: `Initial and destination airports are the same.`
    - Cannot find an existing path from the staring airport to the destination airport: `Cannot find a path!`
  - **Dijkstra Algorithm**: find the shortest path from a starting airport to every other airports in the graph

        - input: `starting airport, destination airport`
        - output: the shortest path from starting to destination and the distance of this path
    - **A* Algorithm**: find the shortest path between two airports with specific airports that we do not want to visit
        - input: `starting airport, destination airport, airports that we want to avoid`
        - output: the shortest path from starting to destination and the distance of this path.
    - The following figure shows the difference between Dijkstra and A* algorithm from CMI to JFK. In the A* algorithm, we add ORD to be the forbidden airport.

```
Shortest Path, with a distance of 1404 km, is: CMI -> ORD -> JFK.
Shortest Path, with a distance of 3348 km, is: CMI -> DFW -> JFK, with
 the following airports avoided: ORD.
Invalid Input!
Initial and destination airports are the same.
Cannot find a path!
```